PROOF OF INVENTION

This document shows the conception of the invention claimed in US patent application 09/498,505 as disclosed in the document

[2] Developing Publish/Subscribe Applications with iBus - Technical White Paper

Revision **1.1**, *Mon May 17 13:58:23 1999 UTC* (4 years, 10 months ago) by *silvano*

which in the following shall be referred to as "White Paper"

A copy of the white paper in which the relevant sections are highlighted is enclosed.

SYNOPSIS OF CLAIMED MATTER AND DISCLOSURE IN WHITE PAPER

The subject matter of the independent claims 1, 8, 11, 12 is disclosed in the white paper. In the following, reference is made to claim 1 alone. The remaining independent claims are directed to a method for running a message system and a computer program product for implementing a message system with the same features as claim 1.

Claim 1, as currently pending, is:

A 1. A messaging system for delivering data in the form of portable message formats between message clients,

B <u>the messaging system based upon a publish/subscribe mechanism or based upon a point-to-point protocol or based upon both a publish/subscribe mechanism and a point-to-point protocol,</u>

C <u>the messaging system</u> comprising at least one transport protocol adapter,

D **whereby** at least one transport protocol is implemented before start-up of the message server and/or is implemented by a code at runtime of the message server

E and said at least one transport protocol adapter

- comprises a logic to interface with said at least one transport protocol,

F - comprises another logic to specify a message delivery quality and

G is pluggable for being started and/or stopped at runtime of the message server.

wherein the underlined text has been added in a response to the office actions, and reference letters A-F denoting the different features are inserted for convenience.

Exhibit C

Disclosure of white paper

The White Paper discloses Softwired's concept for providing a framework on which publish/subscribe applications can be established. In contrast to former applications, the iBus system works transparently over a variety of underlying communication protocols and qualities of service. In order to allow this portability, protocol objects and a quality of service mechanism are introduced, as is summarized in section 5.

The reduction to practice is implied in the downloadable software mentioned on page 12, section 7.

The individual features of the claim are disclosed in the White Paper in the following manner. References to text in the White Paper are denoted by

page number / (sub)section number (if the heading appears on the same page) / line number

Feature A

A messaging system for delivering data in the form of portable message formats between message clients

The fact that a messaging system is the topic of the White Paper is obvious from the white paper as a whole, and stated explicitly in the first sentence:

3/1/1-2 iBus is a pure Java™*publish/subscribe software bus* allowing distributed components to exchange information via a variety of communication protocols and qualities of service.

The portability of message formats regardless of what channel and transport protocol is used is shown e.g. in:

3/1/12-14 It allows iBus to deliver information not only using the Internet protocol family, but also by protocols such as infrared, paging, and satellite communication.

9/5/6-7 Hence the application can be "ported" from the IP network to the satellite network (and back again) very easily.

Feature B

the messaging system based upon a publish/subscribe mechanism or based upon a point-to-point protocol or based upon both a publish/subscribe mechanism and a point-to-point protocol, The different underlying protocols are mentioned in various places, such as:

The title of the white paper itself!

3/1/16-17	iBus supports tying a QoS to each iBus communication channel, i.e. reliable point-to-point communication, reliable multicast,
8/4/1-3	In addition, synchronous request/reply style communication is provided. iBus thus offers the complete functionality of a publish/subscribe software bus plus the main features found in ORBs.

Feature C

the messaging system comprising at least one transport protocol adapter,

Section 5.1 (pages 9-10), with reference to Figure 2, describe the conversion of messages through a linear sequence of protocol converters. In Java, the converters are implemented as software objects, called protocol objects. See in particular, with reference to the left half of Figure 2:

10/-/1-6	A protocol stack represents one QoS and consists of a linear list of *protocol objects*. If an event object is published on a channel, it is passed to the protocol stack of the channel. The event object passes through the topmost protocol object in the stack and is then passed along to the next protocol object in the stack until it reaches the bottommost object. The bottommost protocol object delivers the event to the net-work interface.

This refers to the conversion of a message and its adaptation to the network channel over which the message is transported. This means that the protocol objects are identical in function to the protocol adapters of the application, and that the lower protocol objects (such as IPMCAST) that adapt the message to the transporting channel are identical to the transport protocol adapters of the application.

Similarly, on the receiving side, the right half of Figure 2 represents the receipt of a message through a protocol object (such as IPMCAST, but arranged for the receipt of messages), corresponding to another transport protocol adapter of the application.

10/-/6-7	The event is then received by the bottommost protocol object of the consumer application and is passed up the stack.

Protocol stacks can be modified by exchanging some or all protocol objects, which also implies the adaptation to different transport protocols by means of appropriate transport protocol adapters.

> 11/5.2/1-2 The iBus QoS framework can be extended by implementing new protocol objects. These objects can then be put on top of the existing iBus stacks, or completely new stacks can be composed altogether.

Feature D

> *whereby at least one transport protocol is implemented before start-up of the message server and/or is implemented by a code at runtime of the message server*

The White Paper shows this in:

> 6/2.3/1-2 **The system can be extended gracefully**: you can start as many quote producers and consumers in whatever order you like. It is not necessary to start the consumers before the producers.

An example is given, wherein for tapping into a channel a listener object receiving messages is created. Creating this object, retrieving the protocol adapter class by the JAVA import statement, and executing its software code necessarily implies that the transport protocol for receiving messages is implemented at runtime

> 5/2.2/1-3 An application tapping into the blue chips channel is written to receive stock quotes. The consumer application is structured like the producer application. Its main difference consists in creating a `PublishListener` object which is needed to receive the quotes transmitted on the channel.

Feature E

> *and said at least one transport protocol adapter*
>
> *comprises a logic to interface with said at least one transport protocol,*

This feature is disclosed in the same manner as feature C: The existence of a transport protocol adapter with the function of adapting a message to a specific transport protocol necessarily implies computer code or logic for implementing said function.

Feature F

comprises another logic to specify a message delivery quality and

The message delivery quality is an important aspect highlighted in the White Paper, e.g. in the entire section 5, starting on page 9.

9/5/1-3 The capability to control qualities of services is one of iBus' distinguishing features. This is accomplished by embedding QoS strings into channel URLs. Altering the QoS of transmission channels thus only requires very minor modifications of the iBus application.

And in the introduction:

3/1/12-14 A distinguishing feature of iBus is its *Quality of Service (QoS) Framework*.It allows iBus to deliver information not only using the Internet protocol family, but also by protocols such as infrared, paging, and satellite communication.

Every communication channel can be associated with a quality of message:

8/4/15 The `ChannelURL` class is used to tie a QoS to a channel and will be examined next.

This is done when defining a channel, according to the following syntax:

9/-/2 iBus channel URLs obey the format
`"ibus:"[ QoS":"]"//"[ address][":" address parameter]"/" topic`
iBus URLs always start with the protocol descriptor `"ibus:"`. The next part is an optional *QoS* string indicating the transport protocol and reliability guarantees of the channel.

and further below two examples for specifying QoS as "reliable" or "unreliable"are given

9/-/19-22 `ibus:CRYPT:alias(reliable):///quotes/zrh/bluechips`:Data is encryptedand sent via IP multicast.

`ibus:alias(unreliable):///radiostations/RadioXYZ:` Unreliable multicast is chosen to transmit real-time audio in a best effort manner via UDP.

Feature G

is pluggable for being started and/or stopped at runtime of the message server.

This is mentioned in

6/2.3/3-4 Plug&Play of components: producers and consumers can be relocated from one machine to another at run-time, without need to restart applications.

which means that the producers can be stopped (for relocating to another computer) whereas the message server does not have to be stopped.

Corresponding mechanisms are implemented to handle such cases:

7/3/10 ... whenever a component (producer or consumer) plugs into the channel and whenever a component disconnects.